SAP NetWeaver Application Server, Java^{TM} EE 5 Edition

SAP NetWeaver Scheduler for Java

SAP NetWeaver Application Server, Java^{TM} EE 5 Edition

THE BEST-RUN BUSINESSES RUN SAP

SAP AG
Neurottstraße 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

© Copyright 2005 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes
only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

Disclaimer
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

Documentation on SAP Developer Network
You can find this documentation at
`sdn.sap.com`

Typographic Conventions

Type Style	Represents
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, F2 or ENTER.

Icons

Icon	Meaning
⚠	Caution
💬	Example
💡	Note
🧭	Recommendation
syn	Syntax

Contents

SAP NetWeaver Scheduler for Java

Purpose

The SAP NetWeaver Scheduler for Java provides low-level job scheduling capabilities for applications running on the SAP NetWeaver Application Server, Java™ EE 5 Edition. It is a core service that enables the automated execution of tasks that applications can perform in the background.

For the sake of brevity, in this documentation we refer to the SAP NetWeaver Scheduler Java as the Java Scheduler.

Implementation Considerations

We recommend that you use the Java Scheduler to schedule a moderate number of jobs per day. Although it is possible to run millions jobs in one day, you should limit the number of jobs to much less than 100, 000 per day.

For data integrity and failover reasons, the Java Scheduler persists all information about future, current and past jobs. Due to this, it is heavyweight and is not the best solution for executing several millions jobs a day. If you use the Java Scheduler to execute tasks several times a second, this may overload the system.

For more information about the architecture of the Java Scheduler, see Architecture [Page 2].

Features

The SAP NetWeaver Scheduler for Java provides the following features:

- Developing jobs

  The Java Scheduler implements an object-oriented approach in developing jobs. You develop your own jobs based on message-driven beans to implement the logic of the work to be performed in the background.

- Scheduling jobs

  The Java Scheduler provides time-based job scheduling in that jobs run when a preset date/time condition is fulfilled.

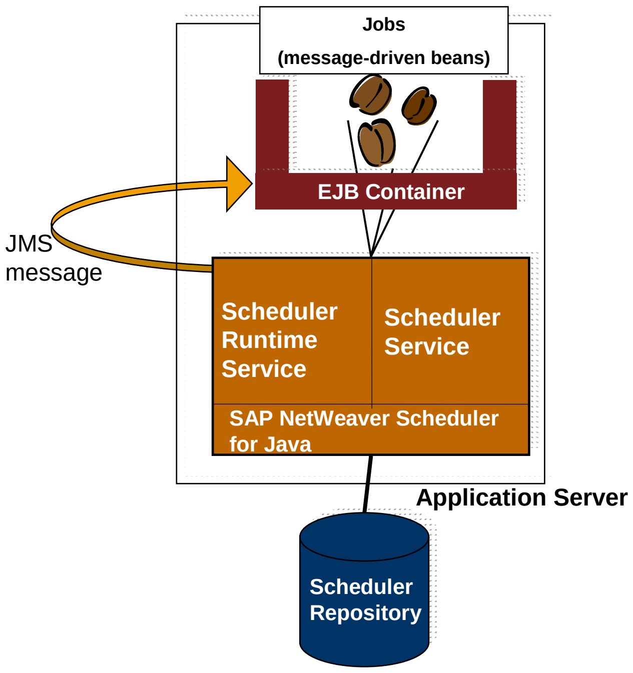
For more information about the basic concepts in job development and scheduling with the Java Scheduler, see Scheduler Job Definition, Job, and Task [Page 4].

For more information about developing jobs, see Developing and Scheduling Jobs [Page 5]. For a step-by-step tutorial on creating and scheduling a job, see Creating and Scheduling Your First Job [Page 23].

1 Architecture

Overview

The architecture of the Java Scheduler is based on two failover-enabled services that control the aspects of job scheduling and execution. Jobs are implemented on the basis of message-driven beans and run in the EJB container. The figure below outlines the architecture of the Java Scheduler.



Services

The Java Scheduler defines two services:

- Scheduler Runtime Service

  Controls all aspects of the runtime of a job. It handles the execution of jobs on the node where it is running, and provides error handling, maintains job definitions and job runtime information, such as job parameters and log files.

- Scheduler Service

  Schedules jobs deployed on the application server and submits them to the scheduler runtime service. The scheduler service accepts requests for rescheduling, canceling and deleting jobs.

The two services persist the complete state of the Java Scheduler in the database.

Service Failover

In cluster environment, the scheduler runtime service is deployed and runs on every cluster node. The scheduler service is also deployed on all cluster nodes but it runs only on a single node at a time. The node where it runs is designated as the singleton node. If the singleton node goes down, the scheduler service gets activated on another cluster node. This mechanism ensures the scheduler service failover.

If a job is running on a node and that node goes down, the execution of the job is resumed on another node. There is no data loss because the complete state of the Java Scheduler is persisted.

Jobs

Jobs are implemented on the basis of message-driven beans. A message-driven bean containing a job is called a JobBean. The execution of JobBeans is handled by the EJB container. A JobBean is executed when it receives a Java Messaging Service (JMS) message from the scheduler runtime service. In cluster environment, the JMS is responsible for load balancing: it decides which JobBean instance on which node gets the request to run.

From a purely scheduling perspective, a job is executed when certain time-based start conditions [Page 11], such as a particular time of the day, are fulfilled. The start conditions of a job are defined when you schedule the job.

For more information about the basic concepts of job definition, scheduler task, and job, see Scheduler Job Definition, Job, and Task [Page 4]

Logging and Tracing

The NetWeaver Scheduler for Java uses the standard SAP Logging framework to log messages on two levels:

- Job level (job logs)

  Logs at job level, or job logs, are logged in the database by every job. The following rules apply for job logs:

  - A job log is always associated with the job instance that logged it. The lifetime of the job log matches the lifetime of the job.
  - The log for a job is deleted when the corresponding job is deleted, for example when the job's retention period has expired.
  - Job logs are not overwritten by a rolling log write strategy. Logs for jobs which are kept for an indefinite period of time cannot be deleted.
  - You can retrieve a log written by a particular job no matter on which cluster node the job ran, or whether the node where the job ran is still part of the cluster.

- Scheduler level (Java Scheduler logs)

  By default, the Java Scheduler logs are logged under the `/System/Server` category, at SYS_SERVER.

Handling the Size of Job Logs

The Java Scheduler allows you to manage the cumulative job log size in the database. A job has a retention period denoting the number of days that a job log is persisted in the database. To prevent database overflow caused by too many job logs, in the job definition's deployment descriptors you can configure a job's retention period. For more information, see Deployment Descriptors [Page 9].

1.1 Scheduler Job Definition, Job, and Task

Definition

The Java scheduler implements an object-oriented approach to developing and scheduling jobs. This section outlines the basic concepts of scheduler job definition, task, and job related to the Java Scheduler.

Scheduler Job Definition

A blueprint representation of a job, also called job metadata, which is deployed on the server. A job definition is not bound to any start conditions, and thus is not a job that is scheduled and ready to run.

Concrete job definition instances, or jobs, are created from a scheduler job definition. An infinite number of job instances can be created from a single job definition.

Scheduler Task

Contains the start conditions and parameter values for a job. A scheduler task instructs the Java Scheduler which job definition to instantiate and run, when, and with what parameters. A scheduler task can trigger a single or multiple instances of a job.

You can create various scheduler tasks based on the same job definition. This allows you to have jobs that are based on the same job definition but run with different parameters and different start conditions.

You have a job definition that backs up data. Out of this single job definition, you can create two scheduler tasks: one that backs up files *MyFile1* and *Myfile2* on every Monday at 8 AM, and another that backs up files *MyFile3*, *Myfile4*, and *MyFile5* every weekend at 6 PM.

Scheduler Job

An instance of a scheduler job definition that is bound to certain start conditions specified in the scheduler task. A job runs once with particular parameter values at a particular point in time or upon a particular event and performs a certain amount of work. One job runs in one thread.

An infinite number of job instances can be created from a single job definition.

The Java Scheduler supports child jobs and job chains.

Use

You use jobs to perform a certain amount of work in the system automatically and in unattended mode. You create a scheduler task based on a job definition to schedule the jobs created from this job definition.

Structure

Scheduler Job Definition

You use the job definition to code the activities that you want an instance of this job definition to perform. A job definition comprises the following:

- Business logic

  The unit of work that is performed when a job instance, that was created from this job definition, runs. You define the job's business logic in the JobBean class.

- Metadata

  Additional information about the job definition, such as name, job parameters description, the JMS resources to be used, and so on. You specify the metadata in the JobBean deployment descriptors.

For more information about the JobBean class and the job definition deployment descriptors, see Job Definition [Page 6].

Scheduler Task

You create a scheduler task in the Scheduler Administrator. A scheduler task comprises the following:

- The name of the job definition you want to schedule.
- Start conditions for the job, which can be of two general types: recurring, and cron. See Scheduler Job Start Conditions [Page 11]
- Values for the job parameters that provide input for the job. For more information about job parameters, see Job Parameter [Page 18].

Scheduler Job

You use a job to have a certain unit of work done automatically and in unattended mode in the system. As a runtime object, a job has a life cycle characterized by job statuses [Page 13].

Integration

You create a scheduler task based on a scheduler job definition. In this way, you instruct the NetWeaver Scheduler for Java to create and execute instances of the job definition (jobs) at the times specified in the scheduler task.

2 Developing and Scheduling Jobs

Purpose

The process below outlines the steps in developing and scheduling jobs for the Java Scheduler in the SAP NetWeaver Application Server, Java™ EE 5 Edition.

Prerequisites

The application server is running.

Process Flow

1. In the SAP NetWeaver Developer Studio, you create a job definition. See Job Definition [Page 6].
2. You deploy the job definition on the application server.
3. You schedule instances of the job definition by creating a scheduler task in the Scheduler Administrator.

   You can create scheduler tasks based on existing job definitions that are already deployed on the server. You can create many scheduler tasks based on the same job definition.

For a step-by-step guide on developing a job definition and creating a scheduler task for it, see the tutorial Creating and Scheduling Your First Job [Page 23].

2.1 Job Definition

Use

Use a job definition to provide a blueprint representation of the job that you want to schedule in the system. In the job definition:

- You implement the business logic that you want an instance of this job definition to perform in the system.
- You provide the metadata for the job definition instances, such as which parameters they take or output, how long the job logs remain in the database, and so on.

Integration

As runtime objects, jobs are created from a job definition, and inherit their business logic from the job definition they instantiate.

Features

Job definitions are implemented after the implementation model of message-driven beans. A message-driven bean containing a job is called a JobBean.

The execution of JobBeans is handled by the EJB container, and a JobBean is executed when it receives a Java Messaging Service (JMS) message from the scheduler runtime service. The NetWeaver Scheduler for Java uses a JMS queue for point-to-point messaging, which allows jobs to be executed asynchronously.

A job definition comprises:

- The JobBean class [Page 7]
- The job definition deployment descriptors [Page 9].

2.1.1 JobBean Class

Definition

Provides an implementation of the business logic that the instances of the job definition have to perform in the system.

Structure

The sample below shows the source code for a JobBean class that logs a *Hello World!* message in the database.

```
package com.sap.scheduler.examples;

import java.util.logging.Logger;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import com.sap.scheduler.runtime.JobContext;
import com.sap.scheduler.runtime.mdb.MDBJobImplementation;
            //Specify message selector
            //and destination type
@MessageDriven(activationConfig={
            @ActivationConfigProperty(
                        propertyName="messageSelector",

      propertyValue="JobDefinition='HelloWorldJob'"),
        @ActivationConfigProperty(
                  propertyName="destinationType",
                  propertyValue="javax.jms.Queue")})
public class HelloWorldBean extends MDBJobImplementation {

      public void onJob(JobContext ctx) {
            //Implement business logic
            Logger log = ctx.getLogger();
            log.info("Hello World!");
      }
}
```

- **Message Selector**

  A JobBean is executed when it receives a Java Messaging Service (JMS) message from the scheduler runtime service. It uses a message selector to restrict the messages it receives from the JMS.

  By using the `ActivationConfig` element of the of the `MessageDriven` annotation, the JobBean specifies the value of the message selector. It has to be in the following format: `JobDefinition = '<Job name>'`.

  `<Job name>` may contain any valid message selector string literal composed of letters, digits, hyphens (-), and underscores (_). `<Job name>` is also specified in the job-definition.xml which is an additional JobBean-specific deployment descriptor. The `<Job name>` value in the JobBean class has to be identical to the one in the *job-definition.xml*.

`<Job name>` specifies the name with which the job definition and its instances appear in the Scheduler Administrator.

The value of `<Job name>` does not depend on the name of the JobBean class. By using this mechanism, you can define two or more different jobs which have different names but use the same implementation.

- **Using a JMS Queue**

  JobBeans are associated with a JMS queue as the destination type for point-to-point messaging, which allows jobs to be executed asynchronously. By using the `ActivationConfig` element of the of the `MessageDriven` annotation, the JobBean specifies `javax.jms.Queue` as the destination type.

- **The onJob() Method**

  The scheduler runtime service provides certain basic services to the job implementation. That is why, the JobBean class cannot implement the `onMessage()` method, which is the standard business method of message-driven beans.

  JobBeans have a single business method, the `onJob()` method. In the JobBean class, you must provide an implementation of the `onJob()` method. The JobBean inherits from an `MDBJobImplementation` base class which itself provides an implementation of the `onMessage()` method.

  If you implement the `onMessage()` method, and not the `onJob()` method, the job definition you deploy on the application server will not be operational.

  In the implementation of the `onJob()` method, you code the logic of the unit of work that the instance of the job definition should perform in the system, when it receives a JMS message from the scheduler runtime service.

- **Job Context**

  As a runtime object, a job accesses the scheduler runtime service through the *JobContext* interface. This interface provides the job access only to runtime services, but no access to scheduling services. You are always authorized to call methods from the JobContext interface.

  Every job has a job context. During runtime, a job object obtains a reference to its job context by an instance of the `JobContext` class. The `JobContext` object is passed as the single parameter to the `onJob()` method as the code sample above shows.

  Through the JobContext interface you can:

  - Execute jobs and retrieve job objects
  - Get a `Logger` object to write job logs to the database
  - Handle job parameters
  - Set a return code for the job

    A job return code is an optional job attribute. It is an integer value output by the job and persisted in the database. A return code is a means for the job to communicate information. The default value of the return code is 0 and indicates that the job completed successfully.

   ○ Work with child jobs.

     You can execute and retrieve child jobs.

The *JobContext* interface is security aware. It allows you to work only with the jobs that were created by the currently logged on user, as well as work with the child jobs whose parent jobs are created by the currently logged on user. You cannot see the jobs created by another user.

For the reference documentation of the *JobContext* interface, you can download the JavaDoc from SAP Developer Network at the Java EE 5 at SAP page → *SAP NetWeaver Scheduler for Java*. For the JobContext interface, see the `com.sap.scheduler.runtime package`.

See also:

Deployment Descriptors [Page 9]

2.1.2 Deployment Descriptors

Definition

Files in XML format that provide metadata for the JobBean. A JobBean has the following deployment descriptors:

- *ejb-j2ee-engine.xml* and *application-j2ee-engine.xml* which are the deployment descriptors for message-driven beans.
- *job-definition.xml* which is a JobBean-specific deployment descriptor.

Structure

- *ejb-j2ee-engine.xml*

  Specifies the destination name and connection factory name which the JobBean uses. The destination name is `JobQueue`, and the connection factory name is `JobQueueFactory`.

  The sample below shows the *ejb-j2ee-engine.xml* deployment descriptor for a *HelloWorld* job definition whose JobBean class is named *HelloWorldBean*.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-j2ee-engine xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <enterprise-beans>
    <enterprise-bean>
      <ejb-name>HelloWorldBean</ejb-name>
      <jndi-name> HelloWorldBean </jndi-name>
      <message-props>
        <destination-name>JobQueue</destination-name>
        <connection-factory-name>JobQueueFactory</connection-factory-
name>
      </message-props>
    </enterprise-bean>
  </enterprise-beans>
</ejb-j2ee-engine>
```

- *application-j2ee-engine.xml*

Incorporates a reference to the APIs of the SAP NetWeaver Scheduler for Java containing the `JobContext` and `MDBJobImplementation` classes. In addition, this deployment descriptor uses a `<modules additional>` element which identifies the application you deploy on the server as a job.

The sample below shows the *application-j2ee-engine.xml* deployment descriptor for a *HelloWorld* job definition, created in an EJB project named *HelloWorldProject*.

```
<?xml version="1.0" encoding="UTF-8"?>
<application-j2ee-engine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <provider-name>sap.com</provider-name>

  <reference reference-type="hard">
    <reference-target provider-name="sap.com" target-type= "service">
           scheduler~runtime
    </reference-target>
  </reference>
  <modules-additional>
    <module>
      <entry-name>HelloWorldProject.jar</entry-name>
      <container-type>scheduler~container</container-type>
    </module>
  </modules-additional>
</application-j2ee-engine>
```

The value of `<entry name>` is the name of the JAR file containing the JobBean class.

- *job-definition.xml*

  An additional deployment descriptor of JobBeans which contains purely job-specific meta information. It specifies:

  - The job name and description

    Free text name and description of the job.

  - The names and properties of job parameters [Page 18].

  - The job default retention period

    The retention period determines the number of days, for which the job logs are persisted in the database.

  The sample below shows the *job-definition.xml* deployment descriptor for a job definition with name *HelloWorld* that only logs a *Hello World!* message and has no job parameters.

```
<job-definitions>
    <job-definition name="HelloWorldJob">
    </job-definition>
</job-definitions>
```

The job definition name has to be identical with the value of the message selector specified in the JobBean class [Page 7].

For more information about the Document Type Definition of the *job-definition.xml*, see Job-definition.dtd [Page 14].

3 Reference

This section provides more information about:

- Scheduler Job Start Conditions [Page 11]

  Provides details about the possible time-based start conditions which you can use when creating a scheduler task.

- Scheduler Job Statuses [Page 13]

  Lists and explains the statuses in the lifecycle of jobs.

- Job-definition.dtd [Page 14]

  Provides the DTD of the *job-definition.xml* deployment descriptor.

- Job Parameter [Page 18]

  Explains the characteristics of job parameters and provides and example.

- Jobs Behavior in Irregular Circumstances [Page 20]

  Explains the behavior of scheduled jobs in situations such as scheduler unavailability at job execution time, change of system time, and so on.

3.1 Scheduler Job Start Conditions

Use

A scheduler job starts to run every time certain start conditions are fulfilled. You use start conditions to schedule jobs and trigger automatic job execution.

Integration

You set start conditions to the instances of a job definition, or jobs, when you create a scheduler task for that job definition in the Scheduler Administrator. The start conditions you set in the scheduler task apply to every job instance created from the job definition.

Prerequisites

To be able to schedule a job, make sure that:

- A job definition is deployed on the server.
- The Java Scheduler is running.

Features

In the Java Scheduler, you can schedule a job to run when a particular time comes (time-based scheduling). The following time-based start conditions are possible:

- Start once or periodically at a particular time (recurrent execution)

  - Start once

    The job runs once at the specified time.

    This is a special case of recurrent execution, in which the job has a single iteration, as opposed to the several iterations in the recurrent periodical start condition.

  - Start periodically

    The job runs once at a particular date and time and then runs recurrently again a specified number of times at specified regular intervals. The recurrent periodical job can run infinitely.

- Start once or periodically on a day relative to the start/end of the month (cron execution)

  With cron start conditions, you define the minute, the hour, the day of week and/ or month, and the year when you want the job to run.

  - Start once

    The job runs once at the specified time.

  - Start periodically

    The job runs at the specified intervals. You can set the job to run one or more times within a period of time that occurs at regular intervals.

Example

The table below provides examples of the various time-based start conditions which you can use to schedule jobs.

Start Condition	**Example**
Recurrent	
• Once	Run the job on December 24 at 6 PM.
• Periodic	• Run the job on Monday October 24 at 10 AM and repeat 10 times every 24 hours. • Run the job on Monday, October 24 at 8AM and repeat every 10 minutes until 6PM on the same day.
Cron	
• Once	Run the job on the first weekend in January in 2010.
• Periodic	• Run the job every weekday at 8:10 AM. • Run the job every weekday, every quarter of an hour between 8 AM and 6 PM in 2010.

3.2 Scheduler Job Statuses

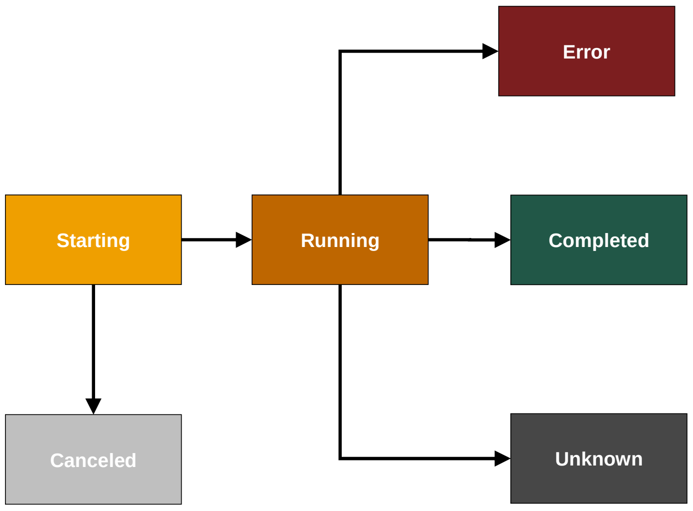
Definition

The status of a scheduler job signifies the job condition at a certain point in the job's life cycle. A job can be only in one status at a time. A job can be in any of the six job statuses outlined in the table below.

Job Statuses

Status	**Meaning**
Starting	The job is currently being started. The status *starting* is possible when a JMS message was sent to trigger the job but the job has not yet received it. This delay in the JMS message receipt is possible if currently there are not enough threads to run a job.
Running	The job is currently performing its unit of work.
Completed	The job has finished its unit of work.
Error	The job has completed its unit of work but threw an exception during execution, or it is clear that the job has failed due to certain problems.
Unknown	The state of the job is not known. The status *unknown* is possible when a node, during its start up, detects that there are jobs currently running on it.
Canceled	The execution of the job was canceled while the job was in status *Starting*.

The figure below shows the possible job statuses and their transitions.



Job Statuses

3.3 Job-definition.dtd

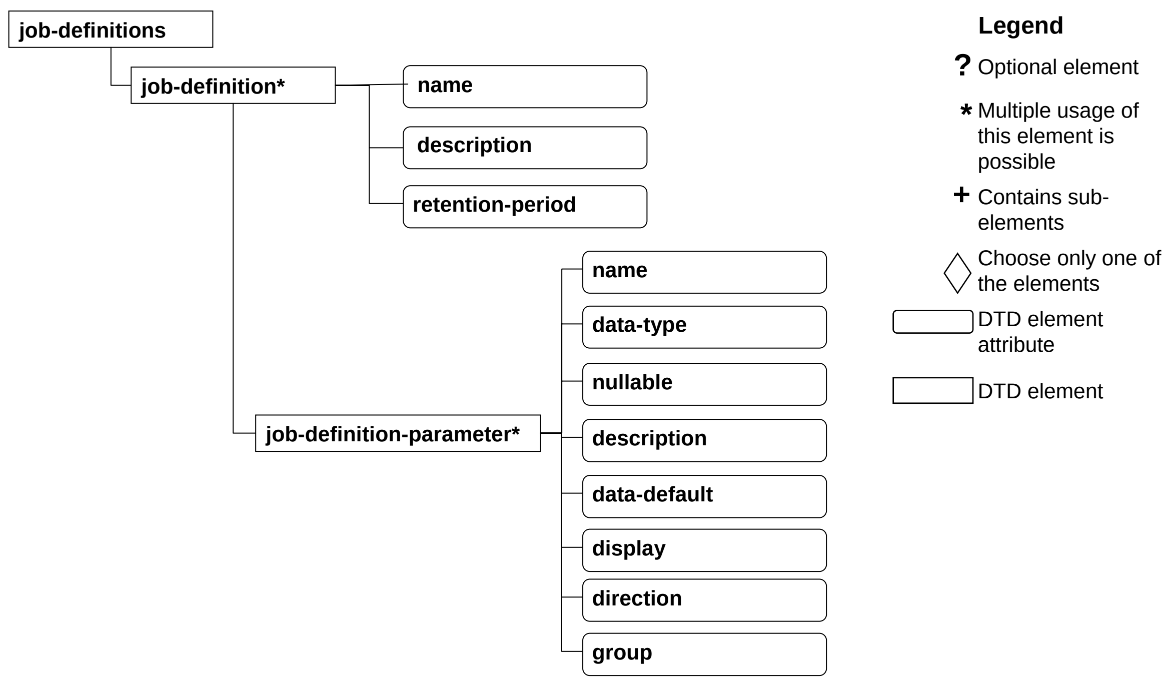
Definition

An XML Document Type Definition that describes how you can specify additional information about job definitions. Every job definition must have a *job-definition.xml* deployment descriptor.

Structure

The *job-definition.xml* has the following structure:

Job-definition.dtd



DTD Description

DTD Element / Attribute	Description
job-definitions	The root element of this deployment descriptor. It contains additional information about one or more job definitions.
job-definition	This element contains additional information about one job definition. Used in: job-definitions. Contains: name, description, and retention-period.
name	The name of the job definition. The value of this attribute has to be the same as the value of the message selector specified in the JobBean class [Page 7]. The job definition is displayed with this name in the Scheduler Administrator. Used in: job-definition.
description	A free text description of the job definition. The description is displayed in the Scheduler Administrator. Used in: job-definition.

retention-period	Determines the number of days, for which the job logs are persisted in the database. The possible values are: • Positive number (n) Keep the job for the specified n days. • 0 Do not keep the job logs • -1 Keep the job logs forever. Used in: job-definition.
job-definition-parameter	This element contains additional information about job parameters. Used in: job-definition. Contains: name, data-type, nullable, description, data-default, display, direction, and group. Used in: job-definition-parameter
name	The name of the parameter. The parameter name you specify here has to be the same as the name you specify for the parameter in the JobBean class. Used in: job-definition-parameter
data-type	The type of data passed by the parameter. The supported data types are: string, float, double, integer, long, Boolean, date, and properties. Used in: job-definition-parameter
nullable	Defines whether the parameter must be specified when a scheduler task is created for the job definition. The possible values are: • Y The parameter has to be specified. • N The parameter does have to be specified. If you do not specify a value, the element takes N as its default value. Used in: job-definition-parameter

description	A free text description of the parameter. The description is displayed in the Scheduler Administrator. Used in: job-definition-parameter.
data-default	The default value that the parameter takes if no value is explicitly specified when a scheduler task is created for the job definition. Used in: job-definition-parameter.
display	Specifies whether the parameter appears in the user interface when the job is scheduled. The possible values are: • `Y` The parameter is displayed. • `N` The parameter is not displayed. Used in: job-definition-parameter.

direction	Specifies whether the parameter is incoming or outgoing for the job. The following directions are possible: • IN The parameter is passed to the job. The parameter value provides input for the job to process. • OUT The parameter is passed from the job. The parameter value is the job’s output. • INOUT The parameter is passed to the job, the job processes it, and then passes it out. The values of all parameters from the execution of each job instance are stored in the database. The parameter values from a job execution are not overwritten with the parameter values from successive job executions. For example, if a job that has OUT parameters runs twice, the values of the OUT parameters for each of the two job executions is persisted. The value of the OUT parameter from the second job execution does not overwrite the value of the OUT parameter from the first job execution. Used in: job-definition-parameter.
group	Job parameters can be grouped in the Scheduler Administrator to improve readability. This property defines the group in which a parameter shows in the Scheduler Administrator. Used in: job-definition-parameter.

3.4 Job Parameter

Definition

An optional attribute of jobs which allows them to get or pass information and thus produce a specific output. A job can have zero, one, or more job parameters.

Use

You use job parameters for the following purposes:

- Pass data to a job. In this case you provide certain input for the job to process.
- Pass data from a job to the outside world. In this case, you get a specific output from the job.
- Pass data between jobs. In this case, you can use job parameters as means of starting job chains.

You declare the job parameters in the *job-definition.xml* and you implement the business logic of the job in the JobBean class [Page 7]. The business logic can access the job parameters. When you create a scheduler task for the job definition, you set values of the parameters that provide input for jobs.

Structure

Job parameters have properties such as name, direction, data type, and so on. You specify these parameter properties in the `job definition parameter` element of the *job-definition.xml* deployment descriptor. For more information about the possible parameter properties and their values, see the Document Type Definition of the job-definition.xml [Page 14].

See also:

Example: Job Parameters [Page 19]

3.4.1 Example: Job Parameters

You have an accounting job named *ProcessPurchaseOrders* with the following abstractly-defined business logic: process customer purchase orders for a period of time and provide the total order value and the number of orders processed.

You want the *ProcessPurchaseOrders* job to process orders for customers whose names start with letters A to M from the alphabet for the month of April 2005.

The table below summarizes the parameter values and properties you have to specify in the job definition:

ProcessPurchaseOrders Job Parameters

Parameter Name	Parameter Value	Data Type	Direction
StartDate	4-1-2005	Date	IN
EndDate	4-20-2005	Date	IN
StartLetter	A	String	IN
EndLetter	M	String	IN
OrdersProcessed	Not Applicable	Integer	OUT
OrderValueProcessed	Not Applicable	Integer	OUT

The *job-definition.xml* file needs to contain the names, data types and direction of the parameters. You specify the parameter values and data types in the JobBean class.

You cannot specify values for the *OrdersProcessed* and *OrderValueProcessed* parameters in the JobBean class. The values for these parameters are the job's output and become available after the job has completed.

The sample below shows how those parameters look like in the *job-definition.xml*:

```
<job-definitions>
    <job-definition name="ProcessPurchaseOrders">
      <job-definition-parameter name="StartDate"
                                data-type="Date"
                                direction="IN"/>
      <job-definition-parameter name="EndDate"
                                data-type="Date"
                                direction="IN"/>
      <job-definition-parameter name="StartLetter"
                                data-type="String"
                                direction="IN"/>
      <job-definition-parameter name="EndLetter"
                                data-type="String"
                                direction="IN"/>
      <job-definition-parameter name="OrdersProcessed"
                                data-type="Integer"
                                direction="OUT"/>
      <job-definition-parameter name="OrderValueProcessed"
                                data-type="Integer"
                                direction="OUT"/>
    </job-definition>
</job-definitions>
```

For more information about the *job-definition.xml* deployment descriptor, see Deployment Descriptors [Page 9]. For the DTD of *job-definition.xml*, see Job-definition.dtd [Page 14].

For an example on working with job parameters, see the tutorial Creating and Scheduling Your First Job [Page 23].

3.5 Jobs Behavior in Irregular Circumstances

This section outlines the behavior of cron jobs (jobs scheduled with cron start conditions) and recurring jobs (jobs scheduled with recurrent start conditions) and the changes that apply to jobs execution times in the cases of execution overlap, time shift, and scheduler blackout as defined below.

Execution overlap

Execution overlap denotes the following situation: JobA1 and JobA2 are successive instances of the same job definition JobA. At the time when JobA2 has to run, JobA1 is still running.

Behavior

In this case, the second job definition instance starts and runs parallel to the first one, that is, JobA2 starts to run although JA1 has not completed. This behavior applies to both cron and recurring jobs.

Time Shift

Time shift denotes the situation in which the system time of the machine where the NetWeaver Scheduler for Java is running is changed back or forward and stretches before or after the scheduled execution times of jobs.

Time Shift Forward

For example, now it is 1:35 AM, you have a job JobABC scheduled to run a quarter to the hour (at 1:45 AM, 2:45 AM, and so on), and you now change the system time to 3:00 AM.

Behavior

When time is shifted forward, the behavior of cron jobs is different from that of recurrent jobs.

- Cron jobs

  With cron jobs, all the execution times that fall into the time-shift frame are skipped. The next execution times after the time-shift are as scheduled relative to the newly set time. In the example above, the execution times at 1:45 AM and 2:45 AM are skipped. The next execution time of the JobABC is at 3:45 AM.

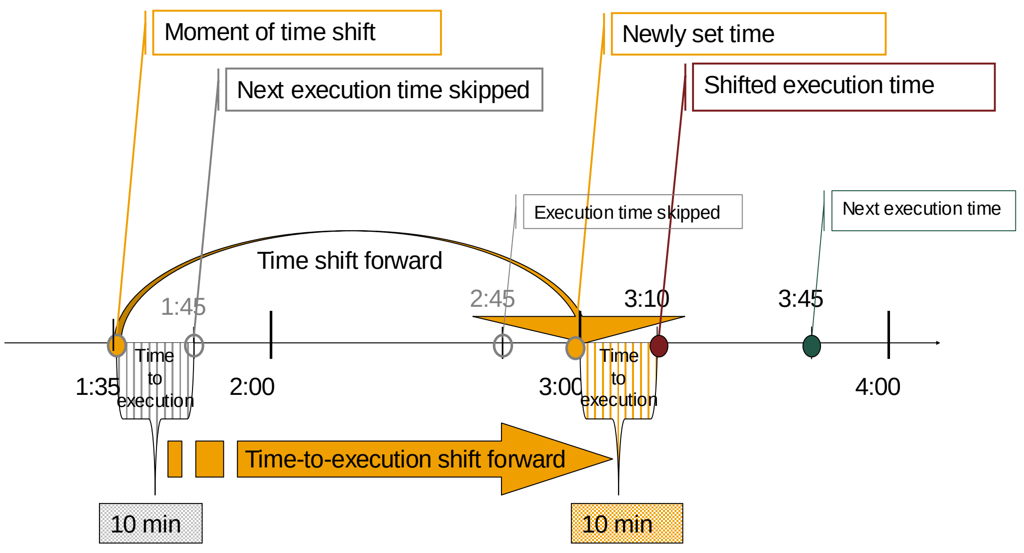
- Recurrent jobs

  With recurrent jobs, all the execution times that fall into the time-shift frame are skipped. In place of all the skipped instances of a particular job definition, a single shifted job runs at a shifted execution time which is relative to the newly set time. The shifted execution time is calculated as follows:

  the moment to which the time is shifted + (the next earliest original execution time of **any** *job waiting for execution – the time at which system time was shifted)*

  The shifted execution time triggers the instances of all job definitions that were skipped. The rest of the execution times after the shifted one remain as scheduled relative to the newly set time.

  In the example above, provided that JobABC is the only scheduled job in the system, the execution times of JobABC at 1:45 AM and 2:45 AM are skipped. In place of the two skipped jobs, a single shifted JobABC instance runs at the shifted execution time of 3:10 AM. The shifted execution time is calculated as 3:00 AM + (1:45 AM – 1:35 AM) = 10 minutes, which makes 10 minutes past 3:00 AM, as the figure below shows.



The time-to-execution is the difference between the next earliest original execution time of **any** job waiting for execution and the time at which system time was shifted.

If apart from JobABC, another job JobXYZ is scheduled to run in the system, for example, 20 minutes to the hour (at 1:40 AM, 2:40 AM and so on), then the shifted execution time is calculated with the next original execution time of the job that runs earliest after the moment of time shift. In this case, this is the execution time of JobXYZ (JobXYZ runs 5 minutes earlier than JobABC).

The shifted execution time is calculated as follows: 3:00 AM + (1:40 AM – 1:35 AM) = 5 minutes, which makes 5 minutes past 3:00 AM. At the shifted execution time, both JobABC and JobXYZ run.

If a recurrent job has a single execution time, and it falls into the time-shift frame, the scheduled execution time of the job is skipped. When the time is changed, the job is executed at a shifted execution time, calculated as described above.

Time Shift Back

For example, now it is 3:35 AM, you have a job JobABC scheduled to run at a quarter to the hour. The job already ran at 1:45 AM, and 2:45 AM. You now change the system time to 1:00 AM.

Behavior

For both cron and recurring jobs, the next execution times are not affected. The jobs do not run again at the execution times that fall into the time changed back. The jobs run again as originally scheduled.

In the example above, the past executions at 1:45 AM, and 2:45 AM that had already taken place are not repeated although the system clock will point to 1:45 AM, and 2:45 AM again. The next execution time of JobABC is at 3:45 AM.

Scheduler Blackout

Blackout denotes the period when the NetWeaver Scheduler for Java is not available at the execution time of a job, for example, if the operating system is suspended, and the job cannot run as scheduled.

Behavior

- Cron jobs

  If the Scheduler Service is stopped, the execution times of the cron jobs that fall into the blackout period are skipped. The next cron job execution time is as scheduled after the Scheduler Service is started again.

  If the Scheduler Service is not available due to garbage collection or communication failure, all cron jobs but one that fall into the blackout period are skipped. A single cron job runs upon system restore.

- Recurrent jobs

  If the recurrent job is scheduled to run infinitely over a regular period of time, that is, no end time and iterations are specified for the job, then the execution times that fall into the blackout period are skipped. Immediately upon service restore, a single instance is executed. The rest of the execution times remain as scheduled.

  If the recurrent job is scheduled to run until a specified time (end time), and the blackout period ends after this end time, a single instance is executed when the system is restored. After this, no more instances of this job definition run.

  If a recurrent job has a single execution time, and this execution time falls into the blackout period, then the job runs when the system is restored.

4 Creating and Scheduling Your First Job

Use

This section guides you by example through the steps you need to complete to develop and deploy a job definition, as well as schedule this job definition from the Scheduler Administrator user interface.

Prerequisites

You are familiar with the following sections:

- Scheduler Job Definition, Job, and Task [Page 4]
- Job Definition [Page 6]

Process

To develop and schedule a job, you need to complete the steps outlined below:

Developing and Deploying the Job Definition

1. Develop the source code of the JobBean class.

2. In the *ejb-j2ee-engine*.xml deployment descriptor, set the correct JMS destination name and connection factory name.
3. Create a *job-definition.xml* descriptor for the JobBean.
4. In the j2ee-*engine*.xml deployment descriptor, set a runtime reference to the APIs of the NetWeaver Scheduler for Java, as well as mark up the job application to identify it as a job.
5. Build the project.
6. Deploy the EAR file on the server.

Scheduling a Job Definition

From the Scheduler Administrator, create and submit a scheduler task for the job definition.

For the step-by-step tutorial on developing and scheduling a job, see Hello Job [Page 24].

4.1 Hello Job

Use

The objective of this tutorial is:

- to guide you step by step through the writing of a job definition of a *HelloJob* that handles parameters and logs messages in the database
- to show you how to create a scheduler task for the job definition.

It is divided into two parts:

1. Creating the Hello Job Definition [Page 25]
2. Scheduling the HelloJob [Page 35]

You can go through the tutorial step by step or download the ready-made job definition of the *HelloJob*.

Using the Ready-Made *HelloJob* Definition

The *HelloJob* described step-by-step in the tutorial is also available for download as a ready-made SAP NetWeaver Developer Studio project which you can import into your own SAP NetWeaver Developer Studio. If you want to go through the complete tutorial later, now you can import the project of the *HelloJob* and go straight to scheduling the *HelloJob* definition.

In this case, proceed as follows:

1. Download and save the *HelloJob_Demo.zip* archive.

   You can download it from the SAP Developer Network at the Java EE 5 at SAP page → *SAP NetWeaver Scheduler for Java.*
2. Extract the *HelloJob_Demo.zip* archive.

   A *HelloJob_Workplace* folder is extracted.
3. In the SAP NetWeaver Developer Studio, open the *J2EE* perspective.
4. Choose *File* → *Import*.
5. In the *Import* window that opens, proceed as follows:
   a. Choose *General* → *Existing Projects into Workspace*, and then choose *Next*.

   b. Next to the *Select root directory* field, choose *Browse*.
   c. Choose the *HelloJob_Workplace* folder you extracted, and then choose *OK*.
   d. Under *Projects*, make sure that *HelloJobProject* and *HelloJobProjectEAR* are selected.
   e. Choose *Finish*.

   The project is imported into the SAP NetWeaver Developer Studio.

6. Set the build path for the project. For more information, see Setting the Build Path [Page 27].
7. Build and deploy the *HelloJobProjectEAR* application. For more information, see Deploying the HelloJob [Page 34].
8. Schedule the *HelloJob*. For more information, see Scheduling the HelloJob [Page 35].

4.1.1 Creating the Hello Job Definition

Scope

In this section you will write the definition of a job that handles parameters and logs messages in the database. You will develop the job in three phases:

1. In a message-driven bean (MDB) class, you code a simple job definition that logs a `Hello` message.
2. You extend the definition by setting two parameters to the job: a parameter for string input whose length the job calculates, and a parameter for the job's output with which the job logs the result of the calculation.
3. You set the correct content of the MDB deployment descriptors and create a *job-definition.xml* descriptor.

Result

At the end, you will have created and deployed the definition for a job that logs the messages: `Hello John` and `The length of your name 4 characters`.

First Step

Creating an EJB Project [Page 25]

4.1.1.1 Creating an EJB Project

Use

This procedure tells you how to create an EJB 3.0 project and an application archive in the SAP NetWeaver Developer Studio.

Procedure

1. In the J2EE perspective, choose *File* → *New* → *Project*.

The *New Project* window opens.

2. In the list of wizards, choose *EJB* → *EJB Project 3.0,* and then choose *Next*.
3. In the *New EJB Project* window that opens, proceed as follows:
    a. In the *Project name* box, enter **`HelloJobProject`**.
    b. Choose Add project to an EAR.
    c. In the *EAR Project Name*, enter **`HelloJobProjectEAR`**.
    d. Choose *Finish*.

Result

HelloJobProject project and the corresponding *HelloJobProjectEAR* application archive are created and appear in the *Project Explorer* pane.

Next Step

Creating a Message-Driven Bean [Page 26]

4.1.1.2 Creating a Message-Driven Bean

Use

Use this procedure to create a message-driven bean in the *HelloJobProject* project, and add the correct build-time reference to the project.

Prerequisites

The *HelloJobProject* project is created.

Procedure

1. In the *Project Explorer*, right click the *HelloJobProject*, and then choose *New* → *Message Driven Bean 3.0*.
2. On the *New Message Driven Bean 3.0* screen that opens, proceed as follows:
    a. In the *EJB Name* field, enter **`HelloJobBean`**.
    b. In the *Default EJBPackage* field, enter **`com.sap.scheduler.examples.hellojob`**.
    c. Choose *Finish*.

Result

The message-driven bean of the *HelloJob* is created.

Next Step

Setting the Build Path [Page 27]

4.1.1.3 Setting the Build Path

Use

The *HelloJob* needs a build time reference to the APIs of the SAP NetWeaver Scheduler for Java. This procedure tells you how to set the correct build path to the *HelloJobProject*.

Prerequisites

The message-driven bean is created in the *HelloJobProject*.

Procedure

1. In the *Project Explorer*, right-click *HelloJobProject*, and then choose *Properties*.

   The *Properties for HelloJobProject* window opens.
2. In the list of properties in the left-hand side pane, choose *Java Build Path*.
3. On the *Libraries* tab page, choose *Add External JARs*.
4. Navigate to *<drive>:\sap\<SID>\<instance name>\j2ee\cluster\bin\services\scheduler~runtime\lib\private* where <SID> is the system ID (for example, JP1), and <instance name> is the instance name of the Java instance (for example, JC00).
5. Choose *sap.com~scheduler~runtime~api.jar*, and then choose Open.
6. Choose *OK*.

Result

The correct build path for the *HelloJobProject* is set.

Next Step

Developing the JobBean Class of the HelloJob [Page 27]

4.1.1.4 Developing the JobBean Class of the HelloJob

Use

Use this procedure to develop the JobBean class of the *HelloJob*.

Procedure

1. From the *HelloJobProject*, open the *HelloJobBean.java* file.
2. Update the source code as shown in the sample below.

   The code excerpt below shows the implementation of a job definition which logs a *Hello* message.

   The `MessageDriven` annotation declares the bean as a message-driven bean. By using the `ActivationConfig` element of the of the `MessageDriven` annotation, you have to further specify `JobDefinition = 'HelloJob'` as the message selector and `javax.jms.Queue` as the destination type.

   The message selector has to be in the following format: `JobDefinition = ‘<Job name>’`

`<Job name>` has to be identical with the name of the job definition which you specify in the job-definition.xml [Page 31]. It can contain any valid message selector string literal composed of letters, digits, hyphens (-), and underscores (_).

The `<Job name>` variable does not depend on the name of the JobBean class. By using this mechanism, you can define two different jobs which use the same implementation.

JobBeans have a single business method which is the `onJob()` method. In JobBeans, the `onJob()` method replaces the `onMessage()` method, which is the standard business method of message-driven beans. Jobs inherit from an `MDBJobImplementation` base class. It provides an implementation of the `onMessage()` method which is declared as final in the base class.

You must not provide an implementation of the `onMessage()` method in a JobBean. If you provide an implementation of the `onMessage()` method, the job definition will not be operational.

```
package com.sap.scheduler.examples.hellojob;

import java.util.logging.Logger;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import com.sap.scheduler.runtime.JobContext;
import com.sap.scheduler.runtime.mdb.MDBJobImplementation;

@MessageDriven(activationConfig={
            @ActivationConfigProperty(
                        propertyName="messageSelector",
                        propertyValue="JobDefinition='HelloJob'"),
        @ActivationConfigProperty(
                  propertyName="destinationType",
                  propertyValue="javax.jms.Queue")})
public class HelloJobBean extends MDBJobImplementation {

      public void onJob(JobContext ctx) {

            Logger log = ctx.getLogger();
            log.info("Hello ");
      }
}
```

The `onJob()` method takes a `JobContext` object as its argument which is an instance of the JobContext interface. As a runtime object, a running job obtains a reference to its `JobContext` and executes with it. Through the JobContext interface a job uses the services of the Scheduler Runtime Service. See Job Definition [Page 6]

The *HelloJob* has to access the Runtime via the JobContext interface to get a logger and write logs.

3. Save the file.

For the reference documentation of the *JobContext* interface, you can download the JavaDoc from SAP Developer Network at the Java EE 5 at SAP page → *SAP NetWeaver Scheduler for Java*. For the JobContext interface, see the `com.sap.scheduler.runtime package`.

Next Step

Extending the JobBean Class with Job Parameters [Page 29]

4.1.1.5 Extending the JobBean Class with Job Parameters

Use

This procedure tells you how to modify the *HelloJob* to pass a parameter to it and make the job output a parameter.

Prerequisites

The *HelloJobBean* is created.

Procedure

1. In the *Project Explorer*, from the *HelloJobProject* open the *HelloJobBean.java* file.
2. Update the source code as shown in the sample below.

   The sample below modifies the MDB class from the previous step. It demonstrates how you can pass a parameter as input for the job and how you create a parameter for the job's output.

   You use the `getJobParameter()` method from the JobContext interface to set a `UserName` parameter to the job that takes a string as input. You use the `setJobParameter()` method from the JobContext interface to set a `NameLength` parameter. The job processes the string input of the `UserName` parameter (calculates the number of characters in the string) and outputs the result of the calculation by logging a message in the database.

   You specify the value of the `UserName` parameter's input later when you create the scheduler task in the Scheduler Administrator [Page 35]. You do not specify the `NameLength` parameter's output anywhere, as it depends on the input of the `UserName` parameter.

   You specify the data type and the direction of the parameters in the *job-definition.xml* at step Creating and Editing the job-definition.xml [Page 31]

```
package com.sap.scheduler.examples.hellojob;

import java.util.logging.Logger;

import com.sap.scheduler.runtime.JobContext;
import com.sap.scheduler.runtime.JobParameter;
import com.sap.scheduler.runtime.mdb.MDBJobImplementation;
import javax.ejb.MessageDriven;
import javax.ejb.ActivationConfigProperty;

@MessageDriven(activationConfig={
            @ActivationConfigProperty(
                        propertyName="messageSelector",
                        propertyValue="JobDefinition='HelloJob'"),
        @ActivationConfigProperty(
                  propertyName="destinationType",
                  propertyValue="javax.jms.Queue")})
public class HelloJobBean extends MDBJobImplementation
{

   public void onJob(JobContext ctx)
   {
        String name;
      JobParameter nameParameter = ctx.getJobParameter("UserName");
      if (nameParameter != null)
      {
        name  = nameParameter.getStringValue();
      }
      else
      {
        name = "Specify a name in the UserName field.";
      }
      Logger logger = ctx.getLogger();
      logger.info("Hello " + name);

      int nameLength = name.length();

      logger.info("The length of your name is " + nameLength + "
characters.");
      JobParameter lengthParameter =
ctx.getJobParameter("NameLength");
      lengthParameter.setIntegerValue(nameLength);
      ctx.setJobParameter(lengthParameter);
   }

}
```

3. Save the file.

Next Step

Editing the ejb-j2ee-engine.xml [Page 31]

4.1.1.6 Editing the ejb-j2ee-engine.xml

Use

Use this procedure to edit the *ejb-j2ee-engine.xml* deployment descriptor.

Procedure

1. From the META-INF folder of the *HelloJobProject*, open the *ejb-j2ee-engine.xml.*
2. Update the file as shown in the sample below.

   You have to specify **`JobQueue`** as the destination name, and **`JobQueueFactory`** as the connection factory name as shown in the code sample below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-j2ee-engine xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <enterprise-beans>
    <enterprise-bean>
      <ejb-name>HelloJobBean</ejb-name>
      <jndi-name> HelloJobBean </jndi-name>
      <message-props>
        <destination-name>JobQueue</destination-name>
        <connection-factory-name>JobQueueFactory</connection-factory-
name>
      </message-props>
    </enterprise-bean>
  </enterprise-beans>
</ejb-j2ee-engine>
```

3. Save the file.

Next Step

Creating and Editing the job-definition.xml [Page 31]

4.1.1.7 Creating and Editing the job-definition.xml

Use

The *job-definition.xml* is an additional descriptor which identifies the message-driven bean as a JobBean.

Use the procedures below to create a *job-definition.xml* file for the *HelloJob* and update its content.

Procedure

Creating a job-definition.xml

1. In the *Project Explorer*, right-click the META-INF folder of the *HelloJobProject*, and then choose *New* → *Other.*

2. On the *New* screen that opens, proceed as follows:
    a. Choose *XML* → *XML*, and then choose *Next*.
    b. Choose *Create XML from scratch*, and then choose *Next*.
    c. In the *File name* field, enter **`job-definition.xml`**, and then choose *Finish*.

Updating the Content of the job-definition.xml

1. In the *Project Explorer*, from the META-INF folder of the *HelloJobProject* open the *job-definition.xml* file.
2. On the *Source* tab page, edit the file as the sample below shows.

```
<job-definitions>
    <job-definition name="HelloJob"
                        description="Logs a string and calculates its
length">
      <job-definition-parameter name="UserName"
                                data-type="String"
                                direction="IN"/>
   <job-definition-parameter name="NameLength"
                                data-type="Integer"
                                direction="OUT"/>
    </job-definition>
</job-definitions>
```

In the *job-definition.xml* file of the *HelloJob*, you have to specify the name of the job definition. It has to be the same as the job name specified in the message selector in the JobBean class. For more information, see Developing the JobBean Class of the HelloJob [Page 27].

In addition, you have to declare the name, data type, and direction of the job parameters used in the *HelloJobBean* class.

With the `UserName` parameter, which is of type string, you provide specific input for the job when you schedule the job. That is why, it has to be declared as a parameter with direction `IN`. With the `NameLength` parameter, which is of type integer, the job provides particular output (in this case, it logs a message in the database). That is why, it has to be declared as a parameter with direction `OUT`. The values for parameter direction are not case-sensitive.

Make sure that in the *job-definition.xml* you specify the correct parameter data types that you set in the JobBean class. If there is a mismatch between the data types of the job parameters from the JobBean class and the *job-definition.xml*, the job will not be fully operational.

Optionally, you can specify a description of the job definition which is displayed in the Scheduler Administrator.

For more information about the Document Type Definition (DTD) of *job-definition.xml*, see Job-definition.dtd [Page 14]

3. Save the file.

Next Step

Editing the application-j2ee-engine.xml [Page 33]

4.1.1.8 Editing the application-j2ee-engine.xml

Use

A JobBean needs a runtime reference to the APIs of the NetWeaver Scheduler for Java which contain the `JobContext` and `MDBJobImplementation` classes. This procedure tells you how to add the runtime reference to the *HelloJobProjectEAR*.

Procedure

1. In the *Project Explorer*, from the META-INF folder of the *HelloJobProjectEAR*, open the *application-j2ee-engine.xml* file.
2. Update the file as shown below.

   You have to add a reference to the Scheduler API, which contains the `JobContext` and `MDBJobImplementation` classes. In addition, with the `<modules additional>` element you identify the application as a job.

   The value of `<entry name>` is the name of the JAR file containing the JobBean class. You can compose the name of the JAR from the name of the EJB project – *HelloJobProject*, and the file extension *.jar*.

```
<?xml version="1.0" encoding="UTF-8"?>
<application-j2ee-engine xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <provider-name>sap.com</provider-name>
  <reference reference-type="hard">
    <reference-target provider-name="sap.com" target-type= "service">
           scheduler~runtime
    </reference-target>
  </reference>
  <modules-additional>
    <module>
      <entry-name>HelloJobProject.jar</entry-name>
      <container-type>scheduler~container</container-type>
    </module>
  </modules-additional>
</application-j2ee-engine>
```

3. Save the file.

Next Step

Deploying the HelloJob [Page 34]

4.1.1.9 Deploying the HelloJob

Use

This procedure tells you how to build the *HelloJobProjectEAR* project and deploy the *HelloJobProjectEAR.ear* application on application server.

Prerequisites

- The application server is running.
- The SAP NetWeaver Developer Studio is connected to the application server.

  To connect the SAP NetWeaver Developer Studio to the application server, proceed as follows:

  a. In the SAP NetWeaver Developer Studio, choose *Window* → *Preferences*.

  b. From the left-hand side pane, choose *SAP AS Java*.

  c. In the *Message Server Host* field, enter the name of the host where application server is installed, for example **`localhost`**.

  d. In the *Message Server Port* field, enter the port of the message server constructed by the formula 36 + NN, where NN is the instance number of the central services instance.

Procedure

1. In the *Project Explorer*, choose *HelloJobProjectEAR*.
2. Choose *Project* → *Build Project*.

   In the *Project* menu, the *Build Project* option is grayed out if on the same menu the *Build Automatically* option is enabled. In this case, the project is built automatically.

   The *HelloJobProjectEAR* is built.
3. Choose *Window* → *Show View* → *Servers*.
4. On the *Servers* tab page, right click *SAP Server*, and then choose *Add and Remove Projects*.
5. From the list of available projects, choose *HelloJobProjectEAR*, and then choose *Add*.
6. Choose *Finish*.
7. If prompted, log on to the application server as Administrator.

Result

HelloJobProjectEAR.ear generated and published on AS Java. You can now create a scheduler task for the *HelloJob* definition.

Next Step

Scheduling the HelloJob [Page 35]

4.1.2 Scheduling the HelloJob

You have successfully developed the job definition of the *HelloJob* and deployed it on the application server. You can now proceed with creating a scheduler task for the job definition.

This section tells you how to:

1. Schedule the HelloJob with the Scheduler Administrator [Page 35]
2. View the output of the HelloJob [Page 39]

4.1.2.1 Scheduling the HelloJob with the Scheduler Administrator

Use

When a job definition is deployed on the application server, you can schedule it by providing start times for the individual job definition instances, or jobs. You can schedule the same job definition to run with various start conditions, for example, both recurrently and cron-like. For more information about the various start conditions, see Scheduler Job Start Conditions [Page 11].

This procedure tells you how to schedule the *HelloJob* by using the Scheduler Administrator.

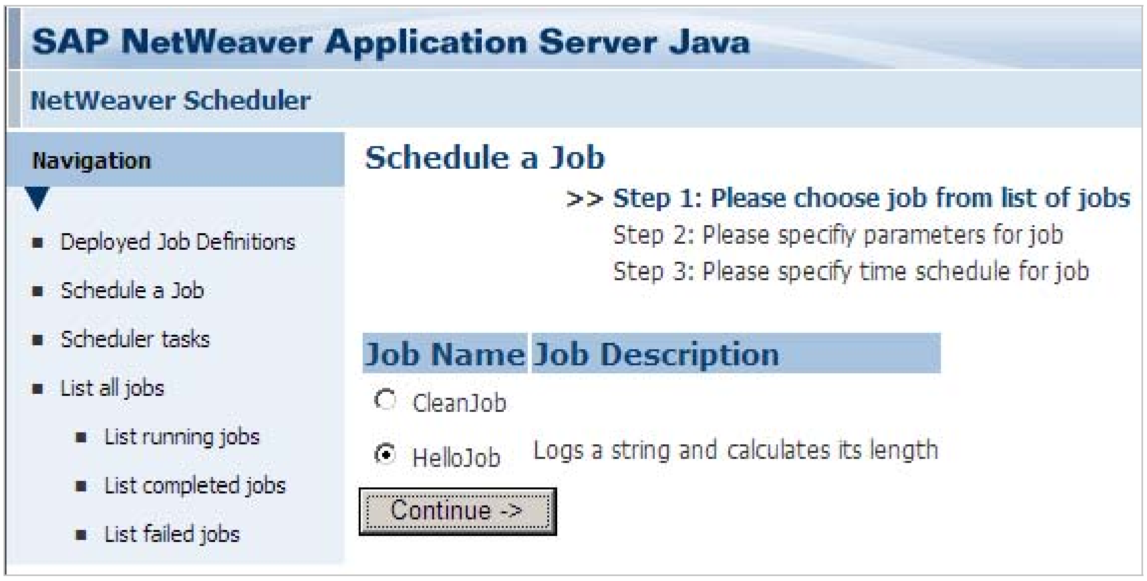
Prerequisites

The application server is running and the *HelloJob* is deployed on it.

Procedure

1. Start the Scheduler Administrator, by browsing **`http://<hostname>:<http_port>/SchedulerAdministrator`**.
   - `<hostname>` is the name of the host where the application server is installed.
   - `<http_port>` is the HTTP port of the application server.
2. If prompted, log on to the application server as Administrator.

   The Scheduler Administrator opens.
3. In the *Navigation* pane, choose *Schedule a Job*.
4. From the list of jobs, choose *HelloJob*.



Schedule a Job

>> Step 1: Please choose job from list of jobs
Step 2: Please specifiy parameters for job
Step 3: Please specify time schedule for job

Job Name	Job Description
○ CleanJob	
● HelloJob	Logs a string and calculates its length

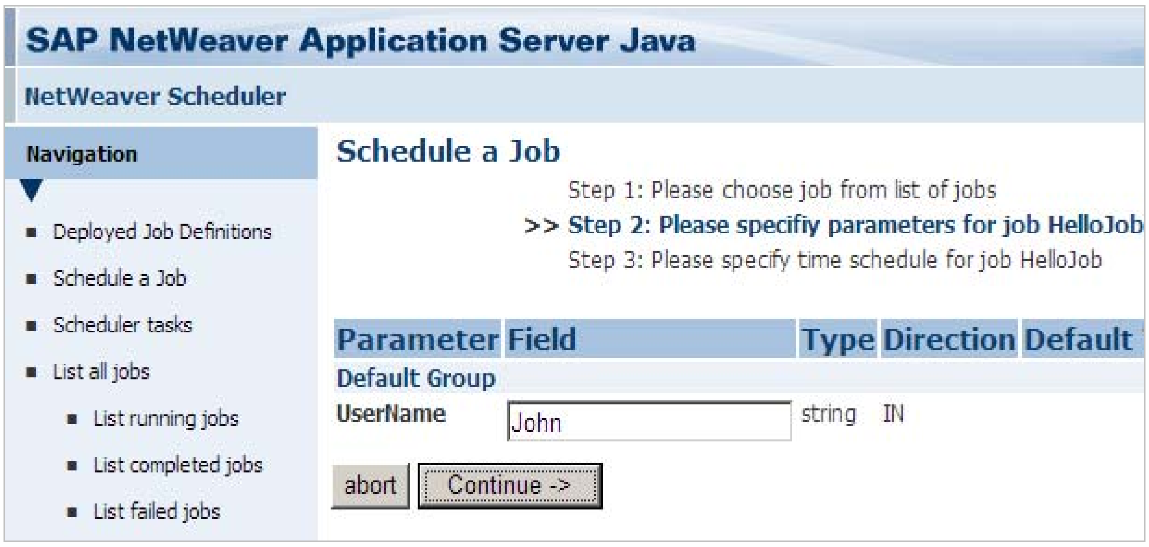
Job Definition Selection

5. Choose *Continue*.

   The screen where you provide input for job parameters opens.

6. In the *UserName* field, enter a value, for example, **John**.

   In this step you provide input for the `UserName` parameter with direction `IN` that you specified in the JobBean class and in the *job-definition.xml* of the *HelloJob* job definition.



Schedule a Job

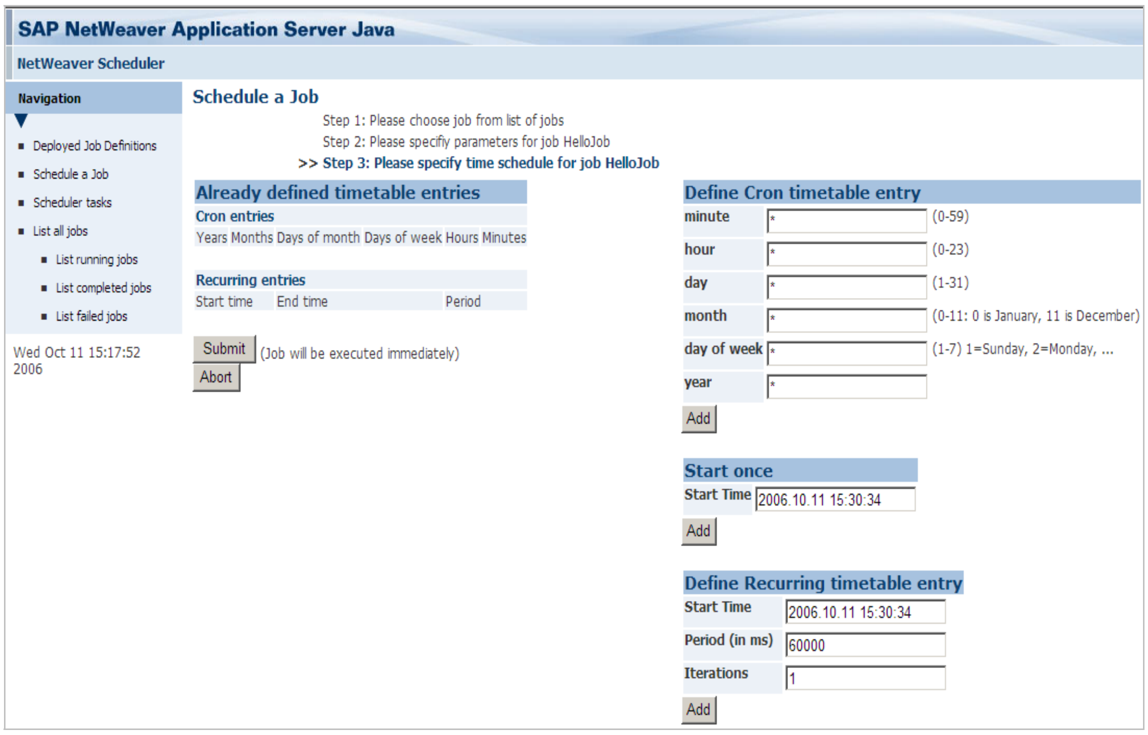
Step 1: Please choose job from list of jobs
>> Step 2: Please specifiy parameters for job HelloJob
Step 3: Please specify time schedule for job HelloJob

Parameter	Field	Type	Direction	Default
Default Group				
UserName	John	string	IN	

Parameter Value for the HelloJob

7. Choose *Continue*.

   The screen where you define the start conditions of the selected job definition opens.


SAP NetWeaver Application Server Java

NetWeaver Scheduler

Navigation

- Deployed Job Definitions
- Schedule a Job
- Scheduler tasks
- List all jobs
  - List running jobs
  - List completed jobs
  - List failed jobs

Wed Oct 11 15:17:52 2006

Schedule a Job

Step 1: Please choose job from list of jobs
Step 2: Please specify parameters for job HelloJob
>> Step 3: Please specify time schedule for job HelloJob

Already defined timetable entries

Cron entries

Years Months Days of month Days of week Hours Minutes

Recurring entries

Start time End time Period

Submit (Job will be executed immediately)
Abort

Define Cron timetable entry

minute	*	(0-59)
hour	*	(0-23)
day	*	(1-31)
month	*	(0-11: 0 is January, 11 is December)
day of week	*	(1-7) 1=Sunday, 2=Monday, ...
year	*	

Add

Start once

Start Time 2006.10.11 15:30:34

Add

Define Recurring timetable entry

Start Time	2006.10.11 15:30:34
Period (in ms)	60000
Iterations	1

Add

Start Conditions

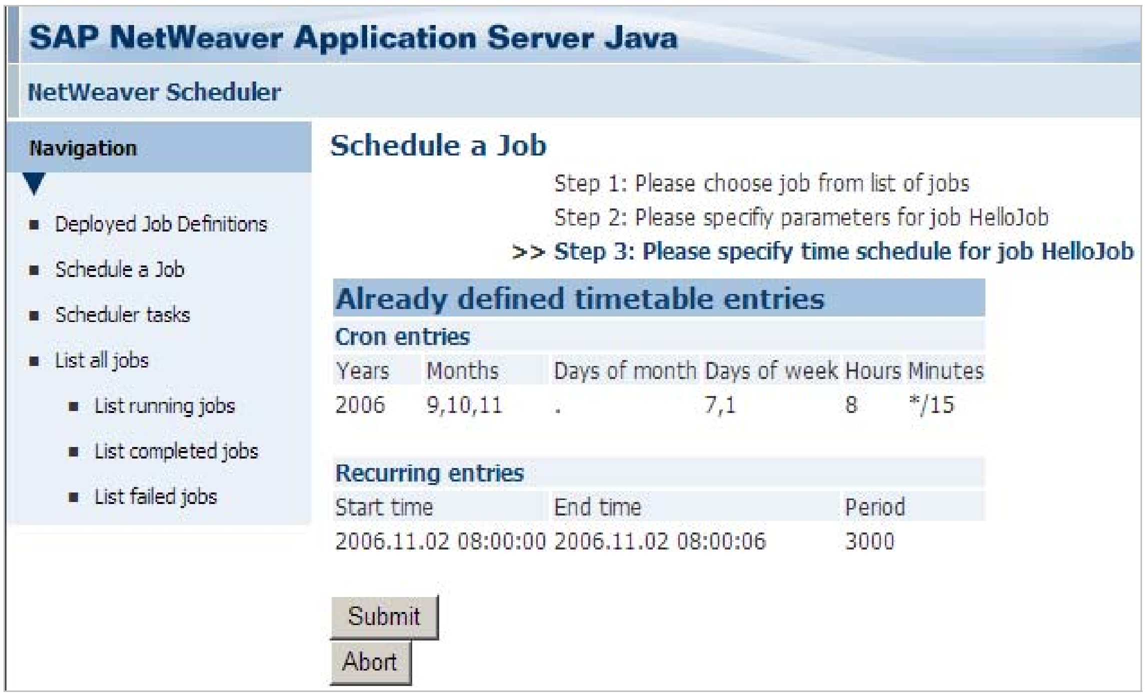
8. Schedule the *HelloJob* by providing the start times for the job instances. The table below shows how you can schedule the job definition to run both recurrently, on the one hand, and cron-like, on the other.

HelloJob Schedules

Start Condition Type	**Procedure**	**Result**
Recurrent execution	Under *Define Recurring timetable entry*, proceed as follows: a. In the *Start Time* field, enter the date and time of the first job execution, for example, at 8:00:00 AM on the next day. In this case, if today is November 1, enter **`2006.11.02 08:00:00`** b. In the *Period* field, enter the period between two successive job executions in milliseconds, for example **`3000`**. c. In the *Iterations* field, enter the number of times you want the job to run, for example, **3**. d. Choose *Add*.	The job will run for the first time on November 2, at 8:00:00 AM, and then run two more times every half a minute.

Cron execution	9. Under *Cron timetable entry*, enter the following values: ○ *Minute* - **`*/15`** ○ *Hour* - **`8`** ○ *Day* - **`*`** ○ *Month* - **`9`**,**`10`**,**`11`** ○ *Day of Week* – **`7`**,**`1`** ○ *Year* – **`2006`** 10. Choose *Add*.	The job will run every quarter of the eighth hour, every weekend in October, November, and December of 2006.

The start times are defined and get listed under *Already defined timetable entries.*


SAP NetWeaver Application Server Java

NetWeaver Scheduler

Navigation
- Deployed Job Definitions
- Schedule a Job
- Scheduler tasks
- List all jobs
  - List running jobs
  - List completed jobs
  - List failed jobs

Schedule a Job

Step 1: Please choose job from list of jobs

Step 2: Please specifiy parameters for job HelloJob

>> Step 3: Please specify time schedule for job HelloJob

Already defined timetable entries

Cron entries

Years	Months	Days of month	Days of week	Hours	Minutes
2006	9,10,11	.	7,1	8	*/15

Recurring entries

Start time	End time	Period
2006.11.02 08:00:00	2006.11.02 08:00:06	3000

Submit

Abort

HelloJob Start Times Definition

11. Under *Already defined timetable entries*, choose *Submit*.

Result

A scheduler task for the *HelloJob* is created. To review details about the task, from the *Navigation* pane, choose *Scheduler tasks*.

Next Step

Viewing the HelloJob Output [Page 39]

4.1.2.2 Viewing the HelloJob Output

Use

This procedure tells you how to open the job log of the *HelloJob* and see the messages that the job logged as its output. You should complete this procedure no earlier than the execution of the first job instance.

Prerequisites

- A scheduler task is created for the *HelloJob*.
- At least one of the instances of the job definition has run.

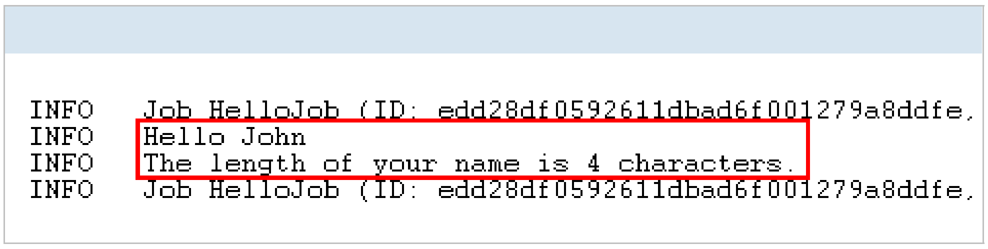
Procedure

1. In the Scheduler Administrator, in the *Navigation* pane, choose *List all jobs*.

   A list of all jobs that are in status RUNNING, COMPLETED, or FAILED is displayed.
2. For any of the entries in the list that have status COMPLETED, under *Actions*, choose *Log*.
3. The job log opens and displays the logged messages:

   Hello John

   The length of your name is 4 characters.



```
INFO    Job HelloJob (ID: edd28df0592611dbad6f001279a8ddfe,
INFO    Hello John
INFO    The length of your name is 4 characters.
INFO    Job HelloJob (ID: edd28df0592611dbad6f001279a8ddfe,
```

Job Output